JavaScript Topics Explained

Import Export (named)

Named exports allow you to export multiple values from a module and import them by their exact names.

```
// math.js
 export const add = (a, b) => a + b;
 export const subtract = (a, b) => a - b;

// main.js
 import { add, subtract } from './math.js';
 console.log(add(2, 3)); // 5
```

Import Export (default)

Default exports are used when a module exports a single value or function.

```
// math.js
 export default function multiply(a, b) {
   return a * b;
 }

// main.js
 import multiply from './math.js';
 console.log(multiply(3, 4)); // 12
```

Aside: array.reduce()

The reduce() method executes a reducer function on each element of the array, resulting in a single output value.

```
const numbers = [1, 2, 3, 4];
 const total = numbers.reduce((acc, num) => acc + num, 0);
 console.log(total); // 10
```

The .reduce() Method with Objects

Using reduce with an array of objects is useful for aggregating values such as totals.

```
const orders = [
  { amount: 10 },
  { amount: 15 },
  { amount: 20 }
];
const totalAmount = orders.reduce((sum, order) => sum + order.amount, 0);
console.log(totalAmount); // 45
```

Default Parameters

Default parameters allow you to initialize named parameters with default values.

```
function greet(name = 'Guest') {
  console.log('Hello, ' + name);
}
greet(); // Hello, Guest
```

Super Challenge Set-up / Solution

This generally involves a large-scale problem requiring multiple JavaScript concepts combined, such as reduce, map, filter, and object handling.

```
// Example setup for a challenge
const people = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 35 }
];

const averageAge = people.reduce((sum, person) => sum + person.age, 0) /
people.length;
console.log(averageAge); // 30
```

The Ternary Operator

The ternary operator is a shortcut for if-else statements.

```
const age = 20;
const status = age >= 18 ? 'Adult' : 'Minor';
console.log(status); // Adult
```

The Rest Parameter

The rest parameter allows you to represent an indefinite number of arguments as an array.

```
function sum(...numbers) {
  return numbers.reduce((a, b) => a + b, 0);
}
console.log(sum(1, 2, 3, 4)); // 10
```

Spread Syntax

Spread syntax allows an iterable such as an array to be expanded in places where zero or more arguments or elements are expected.

```
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];
console.log(arr2); // [1, 2, 3, 4, 5]
```

Short-circuiting with OR (||)

Short-circuiting with || returns the first truthy value.

```
const name = '' || 'Guest';
console.log(name); // Guest
```

Short-circuiting with AND (&&)

Short-circuiting with && returns the first falsy value or the last truthy value.

```
const isLoggedIn = true;
isLoggedIn && console.log('Welcome!'); // Welcome!
```

Switch Statements

Switch statements are used to perform different actions based on different conditions.

```
const color = 'blue';
 switch (color) {
   case 'red':
     console.log('Color is red');
     break;
   case 'blue':
     console.log('Color is blue');
     break;
   default:
     console.log('Unknown color');
 }
```

Constructors: Date()

The Date() constructor creates a new date object representing a single moment in time.

```
const now = new Date();
 console.log(now.toString());
```

The Error() Constructor

The Error constructor creates an error object for custom error handling.

```
try {
   throw new Error('Something went wrong');
 } catch (e) {
   console.log(e.message); // Something went wrong
 }
```

Objects with Methods and 'this'

Inside an object method, 'this' refers to the object itself.

```
const person = {
   name: 'Alice',
   greet() {
     console.log('Hello, ' + this.name);
   }
 };
 person.greet(); // Hello, Alice
```

Objects to Constructor Functions

Constructor functions are templates for creating objects.

```
function Person(name) {
  this.name = name;
}
const bob = new Person('Bob');
console.log(bob.name); // Bob
```

Constructor Function to Classes

Classes provide a cleaner syntax to create objects and handle inheritance.

```
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    console.log('Hi, I am ' + this.name);
  }
}
const alice = new Person('Alice');
alice.greet(); // Hi, I am Alice
```

Debugging: Errors and try...catch

JavaScript provides try...catch blocks to handle exceptions and debug gracefully.

```
try {
  let result = 10 / 0;
  if (!isFinite(result)) throw new Error('Invalid math operation');
} catch (e) {
  console.log('Caught an error:', e.message);
}
```